Building EUROPA

Software Requirements

To begin with, you will need the following software installed on your system:

- Jam 2.5 - An automated build system *(replacement for* `make`*)*
- Java 1.5 - A platform independent programming language and runtime.
- Doxygen - An automated documentation generator. *(required only to generate API documentation)*

The following software is required and tested versions are provided in a ThirdParty? repository (see XXX below for details).

- CPPUnit - A C++ unit testing framework (you may need to rerun ldconfig to get this library installed properly)
- Log4cxx - A C++ logging package that emulates the popular Log4j system.

The following software is also required, but is likely already installed on your system:

- Subversion - version control system.
- GCC 3.3+ - GNU Compiler Collection.
- Perl - A general purpose scripting language, used for some utility scripts in PLASMA. *(not strictly required)*
- SWIG 1.3.36+ - Tool that allows us to expose C++ interfaces in Java and other languages.
    - mac users : if you download it from the swig site, make sure you get swig, not macswig, which is a very old version available from that site)
    - version 1.3.29+ will work for everything except one small bug (see #169)
- Cygwin - A POSIX environment for Microsoft Windows.
- NOTE: For Windows, do NOT check out PlanWorks (only PLASMA).

The following software is included in Europa, but you may wish to install your own version or already have your own version installed:

- ANT - The top-level build tool for Europa

Building EUROPA

Check out the source code

EUROPA's source code is kept on a subversion code repository.

The Bleeding Edge

To get the most recent version of the code, run (PlanWorks contains the PlanWorks and PSUI visualization tools, while PLASMA contains the core EUROPA software) :

Note : If you're an authorized EUROPA committer, you must checkout your code using https instead of http or you will not be able to commit. You'll be asked for you googlecode credentials before the checkout operation can proceed. If you don't happen to know your password (which is automatically generated and different from your gmail password), then you should sign into your gmail account and visit http://code.google.com/hosting. Then select your profile and click the settings tab.

```
svn co http://europa-pso.googlecode.com/svn/PlanWorks/trunk PlanWorks
svn co http://europa-pso.googlecode.com/svn/PLASMA/trunk PLASMA
```

Windows Users:: Do NOT check out PlanWorks, as it is not supported, and confuses EUROPA compilation.

Official Releases

To get an officially released version of EUROPA, run:

```
svn co http://europa-pso.googlecode.com/svn/PlanWorks/tags/<release_tag> PlanWorks
svn co http://europa-pso.googlecode.com/svn/PLASMA/tags/<release_tag> PLASMA
```

Tags for official EUROPA releases (click on the tag to see details) :

Tag	*Date*
EUROPA_2.1.2	2008/08/12
EUROPA_2.1.1	2008/01/11
EUROPA_2.1.0	2007/11/05

Configure your environment

The following environment variables are needed to build and run EUROPA (shown here added to `~/.bashrc`, assuming that PLASMA was checked out in the $HOME/svn directory):

```
export PLASMA_HOME=$HOME/svn/PLASMA                                  # PLASMA_HOME is only used to ease the
export ANT_HOME=$PLASMA_HOME/ext/ant                                 # EUROPA includes a version of Ant with
export PLANWORKS_HOME=$HOME/svn/PlanWorks                            # wherever you checked out PlanWorks

# On Cygwin, the following two paths should be defined with Windows style paths without spaces.
export JAVA_HOME=/usr/java/jdk1.5.0_12                               # the directory where you installed Jav
export EUROPA_HOME=$PLASMA_HOME/dist/europa                          # where the results of the build are pl

export LD_LIBRARY_PATH=$EUROPA_HOME/lib:./build/lib:.                # DYLD_LIBRARY_PATH on a Mac
```

```
export DYLD_BIND_AT_LAUNCH=YES                                    # Only needed on Mac OS X
```

When you compile jam, you may have to move the 'jam0' executable to 'jam'. You may also need to add the 'jam' ,'ant' and 'swig' executables to your path:

```
export PATH=$PATH:$ANT_HOME/bin:$HOME/programs/jam:$PROG/swig-1.3.36
```

Finally, for reasons we can't fathom, you may need to add the following if you want to use PlanWorks:

```
export OSTYPE=$OSTYPE
```

To see if you have the necessary software requirements, run (note that you must be in the 'bin' directory):

```
% cd <path-to-root-PLASMA-directory>/bin
% checkreqs
```

Build a EUROPA distribution

To build EUROPA, simply run (Note that this will take a while):

```
% cd $PLASMA_HOME
% ant build zip-dist
```

the EUROPA distribution will be available at $PLASMA_HOME/dist/europa.zip, you can the take the distribution and install it

If desired, Doxygen API documentation can be run by following the directions here.

Advanced Build Configuration

Additional Arguments to Jam

Arguments to the jam exec can be passed through the jam.args variable. This is useful to enable SMP support on multiprocessor machines; for example:

```
  % ant -Djam.args="-j2"
```

will tell jam to use two shells for processing. Jam will take care of dependencies for the compilation. Warning -- do not use this if you are debugging dependency issues!

Common Targets

Here are the most common ant targets to build/test EUROPA components (all are available in $PLASMA_HOME/build.xml):

Target	*Description*
build	Builds all the EUROPA modules
test	Runs all the regression tests

dist	Gathers all the elements of a EUROPA distribution under $PLASMA_HOME/dist/europa
zip-dist	Same as *dist* then puts the distribution directory in a zip file

TODO: document some of the jam targets as well?

Build Options

EUROPA can be built using a few variations. The variations and parameters of the build can be configured through environment variables or parameters to jam. EUROPA supports building one or more "variants" at the same time. The valid variants are (see here for implementation details):

- DEV - unoptimized, unprofiled, debug (all error detection on). The default option, but not necessarily the best.
- OPTIMIZED - optimized, unprofiled, no debug. Error detection is disabled to improve runtime performance; compile times are somewhat slower.
- PROFILE - unoptimized, profiled, no debug. Error detection is disabled.
- COVERAGE - compiles with the flags required by the gcov code coverage tool. See Analyzing Code Coverage below for more information.
- CUSTOM - special variant that examins some environment variables to set the compilation flags.

For example, to build using optimized compiler settings :

```
% cd $PLASMA_HOME
% ant -Djam.variant=OPTIMIZED
```

The output files are named differently for each variant, which makes it possible to build different variants at the same time. It is also not necessary to clean the tree when switching variants. Here are the naming patterns for each of the main variants:

- DEV = <filename>_g<ext>
- OPTIMIZED = <filename>_o<ext>
- PROFILE = <filename>_p<ext>

The fourth variant, CUSTOM is a special variant that examines some environment variables to determine the compilation flags. These environment variables are:

- EUROPA_DEBUG - use debugging
- EUROPA_DEBUG_MESSAGE - enable DebugMsg
- EUROPA_OPTIMIZED - use optimization
- EUROPA_PROFILE - use profiling
- EUROPA_WARN - use additional warnings

In the case of the CUSTOM variant, the five variables are used to construct the name by composition. E.g., the OPTIMIZED variant is identical to the CUSTOM EUROPA_OPTIMIZED variant. See this file for details.

The default variant is DEV, which includes EUROPA_DEBUG_MESSAGE.

Configuring Build Libraries

The EUROPA build supports building modules as different kinds of libraries. The building of modules as different kinds of libraries is also controlled through environment variables or parameters to jam. EUROPA supports building 1 or more library kinds at the same time. The valid library kinds are:

- STATIC - using static compile time linking (*.a)
- NONE - avoid using libraries (link *.o directly) *(This option is no longer officially supported, although it should still function)*
- SHARED - using shared compile time linking (*.so) (dyanic libraries on OS X)
- NONE - Compile without generating libraries at all, all code is linked into executables.

For example, to build without using libraries :

```
% cd $PLASMA_HOME
% ant -Djam.libraries=NONE
```

When building shared libraries, it is necessary to build using position independent code. As a result, the object files are named differently in this case:

- SHARED = <variant>_pic.o
- STATIC/NONE = <variant>.o

Also, depending on which kind of libraries are used, the target executable names are also different.

- STATIC = <filename>.exe
- NONE = <filename>_no.exe
- SHARED = <filename>_rt.exe

The default library kind is SHARED. The library kind NONE is only recommended for situations where cyclic dependencies exist between modules. The library kind SHARED is often the most space efficient. When running with SHARED it is required to add the EUROPA shared library directory to your library search path. The relevant variable depends on your platform, and the way to set it depends on your shell. Here is an example for the sh and bash shells: % LD_LIBRARY_PATH="$LD_LIBRARY_PATH":/home/me/PLASMA/lib % export LD_LIBRARY_PATH where you should replace '/home/me' with the full path to the directory containing your EUROPA checkout from SVN. For csh and tcsh and similar shells: % setenv LD_LIBRARY_PATH="$LD_LIBRARY_PATH":/home/me/PLASMA/lib

Analyzing Code Coverage (*Linux platform only*)

EUROPA's code coverage tool (based on gcov) is used to identify the percentage of the code base that is exercised by the module tests. This is an important quality metric as the higher the percentage of the code base tested the more likely it is that bugs will be detected.

Go to the PLASMA/bin directory and invoke the coverage script to generate the coverage data. The script will run for approximatly 30 mins as it builds a coverage variant of the EUROPA system and runs all the module tests before generating the gcov code coverage data. The script produces the following files in the PLASMA directory.

- coverage-results.txt lists the percentage of each source file executed by Europa's system tests.
- A filename.cc.gcov or filename.hh.gcov file is provide for each source file in the system. It lists the code within the file annotated with the number of times each line has been run. Mulitple pound signs (#) indicate that a line was not executed.
- *.bb, *.bbg, *.da are working files prouced by gcov. They are of no interest to the human reader.

Examples

To build an optimized version with static libraries

```
% cd $PLASMA_HOME
% ant -Djam.variant=OPTIMIZED -Djam.libraries=STATIC
```

To build a debug version with shared libraries

```
% cd $PLASMA_HOME
% ant -Djam.variant=DEV -Djam.libraries=SHARED
```